Machine Learning
&
Deep Learning
-
Made EASY

A perfect guide for your AI enthusiasm

Written by:

Roudayna Zorg & Aziz Souiai

PART ONE

TABLE OF CONTENTS

Intro to AI

"Predicting the future isn't magic,
it's artificial intelligence"
-Dave Watters

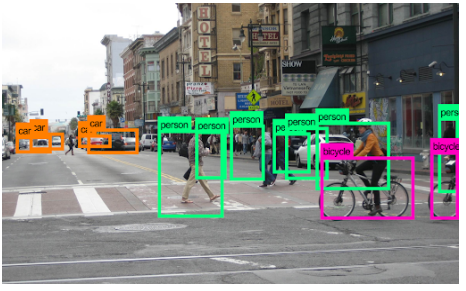
1.1. Defining AI:

John McCarthy, widely recognized as one of the godfathers of AI, defined artificial intelligence as:

"*The science and engineering of making intelligent machines.*"

Artificial intelligence is simply trying to make machines **mimic** human behavior. Humans have specific "powers": humans can see, talk, hear, move, jump… Any application that makes machines mimic human powers is an application of Artificial Intelligence.

This is an example of an application that gives a machine the ability to **'see'** as humans. Thus, this is an **AI** application. In fact, the applications that make a machine '*see*' are **Computer Vision** applications which is a subset of AI we'll see later in this book.

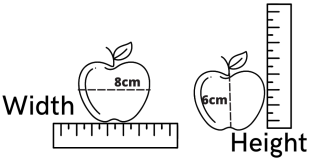


1.2 Rule-Based Approach

Let's suppose we have a dataset that contains images of apples, bananas, melons, and strawberries. We want to make an algorithm to determine whether each image is an apple or not.

A traditional way to approach this problem is by using the **Rule-Based Approach** which is determining a specific set of rules to solve my problem. For our example, we want to predict whether a picture is an apple or not. We can define some set of rules such as:

- Checking the width and height of the fruit.
- Checking the color of the fruit
- Checking the shape of the fruit.



We can then write our algorithm to make our prediction this way:

```
if (width>6 and width<10) and (height>4 and
height<8)
and (shape = 'spherical') and (color = 'green' or
color ='red'):
        print("This is an apple!")
else:
        print("This is not an apple")
```

In the Rule-Based approach, rules are pre-defined. We give our algorithm the input (which is the fruit image in the example above), and the set of rules (width, height, color, shape…). Then, the algorithm gives us the desired output (apple or not).

The figure below illustrates the rule-based approach:

1.3 Machine Learning Approach

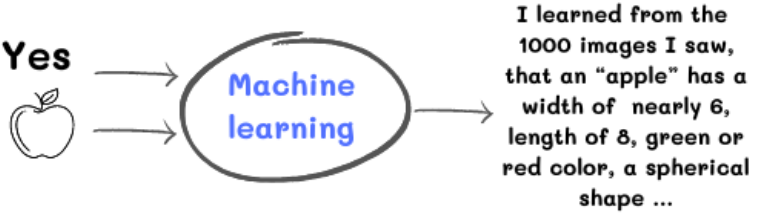
Let's take the same dataset of fruit images and our same goal is to predict whether an image is an apple or not.

Instead of using the traditional way and defining the sets of rules, we can make the machine define those rules for us. That is called **Machine Learning**.

We don't have to worry anymore about defining those rules. Don't waste your time looking for the average width of an apple, its color, its shape, or anything. The machine will do it all for you. Even though it seems magic, believe me, it's not (we'll see how in the next few pages).

Instead of defining the rules, we feed the 'model' (where the magic happens) each image with its appropriate target/answer ('Yes': if it's an apple, 'No': if it's not). The machine will determine the set of rules by itself.

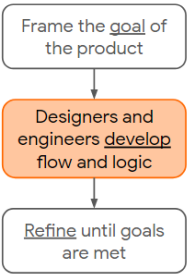
If we want to better understand the figure above, here is an example figure to better illustrate the situation based on the dataset we're working on:



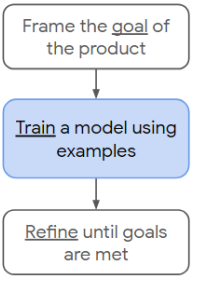
1.4 Rule-Based vs Machine Learning:

The biggest drawback of the rule-based approach comes in situations where we have complex problems that we can't easily define their rules. For example, there are some apples that are much bigger (or smaller) than a regular apple; this might lead to an incorrect classification and lead the developer to work a lot on defining accurate rules. Machine Learning facilitates a lot of the process in this case and provides an easier solution by defining the rules for us.

Rule-Based approach



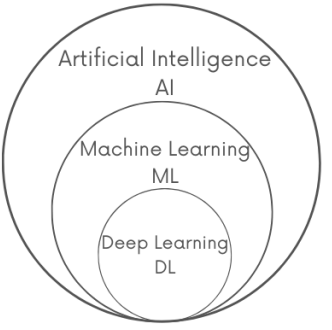
Machine Learning



1.5 AI vs ML vs DL:

At this point, you can define AI and ML by yourself.

- AI is any technique that enables machines to mimic human behavior.
- ML is simply a technique to learn from data.



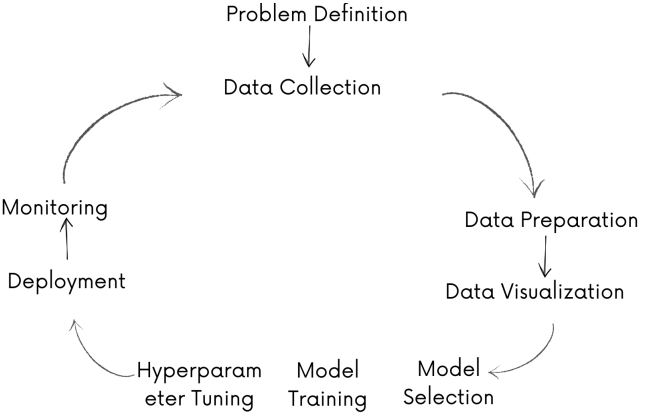
You have for sure encountered the image below if you ever wanted to learn AI:

In fact, it explains that ML is a subset of AI.

Also, it introduces a new field called **Deep Learning (DL)** which we will discover in a future book part. For now, we can admit that DL is a subset of ML that introduces a new concept called neural networks that reflect the same neural networks we have as humans, but of course, much simpler.

ML Life Cycle

The Machine Learning Life Cycle is a systematic process for developing and deploying machine learning models. In the next few parts, we'll cover all the necessary steps in this cycle.



2.1 Problem Definition Understanding:

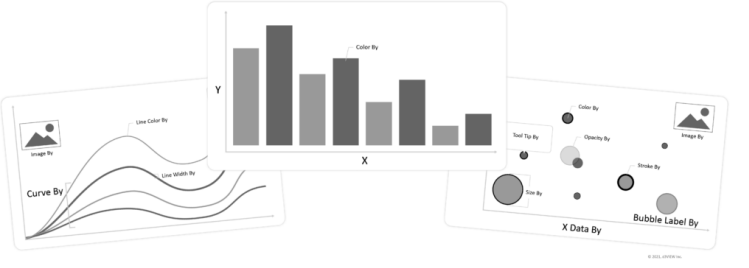
This is the initial step where the problem to be solved is defined. Then we need to understand that problem and gather its requirements. For example, the problem definition for our apple image prediction scenario would be to build a model that can accurately predict whether an image contains an apple or not.

2.2 Data Collection and Preparation:

In this step, data is collected from various sources and prepared for further analysis. ***For example***, in our apple image prediction scenario, we might collect a dataset of images of apples and non-apples, which will be used to train and evaluate our model. The collected images might need to be pre-processed (it might include data cleaning, data transformation … We'll cover all these concepts in the next part of the book) to resize them to a uniform size and format, and any irrelevant or missing information might need to be cleaned. The final dataset will be divided into training, validation, and testing sets to be used in the subsequent steps of the ML life cycle.

2.3 Data Exploration and Visualization:

In this step, data is analyzed and visualized to gain insights and understand its distribution, patterns, and relationships. ***For example***, we might create histograms, scatter plots, and correlation matrices to visualize the distribution of stock prices and the relationship between different variables.



2.4 Model Selection:

This step involves selecting the appropriate model type based on the problem definition, data, and evaluation metrics. ***For example***, we might choose a Convolutional Neural Network (CNN) for our apple image prediction problem. You may ask what CNNs are. CNN is a type of DL model that is well-suited for image classification tasks.

2.5 Model Training and Evaluation:

This step involves training the selected model on the prepared data and evaluating its performance using specific evaluation metrics. ***For example***, we might train our CNN model on 80% of the image dataset and evaluate its performance using accuracy on the remaining 20% of the data. For now, consider training the model as teaching a baby what an apple is: you'll keep giving him apples to discover them and what they look like at a certain point where he gets ready to differentiate an apple from any other fruit.

2.6 Hyperparameter Tuning:

In this step, the parameters of the selected model are optimized to improve its performance. ***For example***, we might tune (change for the better) the number of layers in our CNN to improve its performance on the evaluation data. It's like we're changing the baby's learning parameters: changing the learning environment so the baby gets more

motivated to learn, changing the number of times we give the baby the ***same*** apple so he learns from it…

2.7 Deployment and Monitoring:

Deployment and Monitoring: This is the final step where the model is deployed in a production environment and monitored for its performance over time. ***For example,*** our apple image prediction model might be deployed as a mobile app or on a website and monitored for its accuracy and performance on new images. The model might need to be fine-tuned based on the results of the monitoring and new data, to maintain its performance over time.

Machine Learning Types

Before we move forward and work on each step above, we need to understand the different types of Machine Learning systems.

3.1 Supervised Learning:

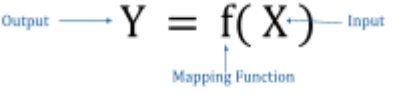
Supervised Learning is the process of training a model on **labeled data** (data containing input and output or also known as features and **labels**).

We can illustrate it as a teacher-student case, where a teacher gives his student 2 main things:

- An input: an apple so the students learn and discover it.
- An output: the label "apple" where the teacher tells his student that that is an apple.

The student thus makes a relation between the input and output. Technically speaking, we say that the student is learning the pattern between the input and the output.

We will tend to model these ***hidden patterns*** by a mapping function as follows:



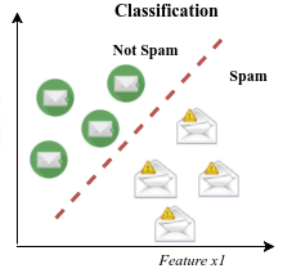
The are 2 kinds of supervised learning problems:

3.1.1 Classification:

Classification is a type of supervised learning where the label we want to predict is categorical (discrete). In other words, we will predict the category the data belongs to (the label belongs to a predefined list of possibilities).

Example of classification problems:

- Image classification: classifying an image into categories such as "apple", "banana", "peach", etc.
- Medical diagnosis: classifying a patient as having a certain disease or not such as "diabetic" or "not diabetic".
- Email spam detection: classifying an email as "spam" or "not spam".
- Sentiment analysis: classifying a piece of text as "positive", "negative", or "neutral"



The mapping function in a classification case will serve to predict in which class our input belongs.

3.1.2 Regression:

Regression is a type of supervised learning where the label we want to predict is numeric (continuous). In other words, we'll predict a numeric value based on previously observed data.

Examples of regression problems:

- Predicting your exam scores based on your study hours: using the relationship between the number of hours spent studying and exam scores to estimate a student's exam score.
- Predicting a person's height based on his/her weight.
- Housing price prediction: predicting the price of a house based on its many features such as location, size, number of bedrooms, number of floors, etc.

3.2 Unsupervised Learning:

Unsupervised Learning is the process of training a model on **unlabeled data** (the data does not contain **labels**). Unsupervised is based on the concept of letting the machine learn the hidden patterns in unlabeled datasets on its own.

We can illustrate it by giving a student a basket of different types of fruits and letting him discover all the fruits and learn their pattern, of course without teaching him or telling him what type is the type of fruit each time. That's why it's called unsupervised since the kid learns by himself.

Here, the kid can have 2 options:

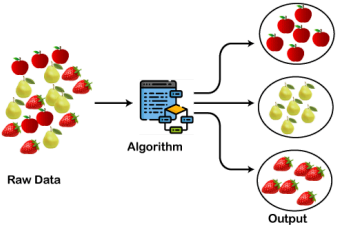
- He can group up different types of fruit based on similarities between those fruits. That's called clustering. Example: grouping up all the fruits that have the yellow color together, grouping up all the fruits that have a heavy weight together …
- He can make a certain association between fruits on certain rules. That's called Association. Example: He

can associate fruits based on his personal preferences, such as

'Apple' → 'Melon' → 'Strawberry' → 'Apple' …

3.2.1 Clustering:

The clustering concept is about grouping data into clusters (groups) where elements in the same cluster are more similar to each other than to those from different clusters.



In other words, clusters are groups of elements (data points) that have feature similarities.

In case you wonder what's the difference between *Clustering* and *Classification* from the image above, here is a small reminder for you:

- Classification case comes when we give the model the data associated with its labels: for each fruit we feed to the algorithm, we feed him with its label too, containing that specific fruit name.
- Clustering case comes when we give the model the raw data directly without specifying what type of fruit each image is and letting the algorithm differentiate between all fruits based on similarities between images.

Some common algorithms for clustering we'll see in this book are K-means and Hierarchical Clustering.

3.2.2 Association:

Association Rules are the process of exploring the data and extracting rules within that data. It discovers relationships and patterns within the data based mainly on statistical methods.

Examples of rules within the data:

- What items from a market basket are bought together.
- What songs from a Spotify playlist are chosen together.

As you may notice, association rules might be used to build some recommender systems, as well as in market basket analysis.

Recommendation systems: *YouTube*, *Instagram*, and *Spotify* are applications of association rules. It's used to figure out which songs, videos, reels are often chosen together. Later, we can recommend to new users new videos, songs… based on the rules defined.

One common algorithm for association rules is the Apriori Concept which is a statistical algorithm that we'll cover later in the book.

3.3 Semi-supervised Learning:

Semi-supervised learning is a type of Machine Learning where the training data includes both **labeled** and **unlabeled** data. The goal is simply to use the labeled data to build a model to predict the labels of the unlabeled data.

This branch of Machine Learning can be useful in cases where acquiring labeled data is either expensive or time-consuming, meanwhile, large amounts of unlabeled data can be readily available.

3.4 Reinforcement Learning:

Reinforcement learning is a type of machine learning where an agent learns to make decisions by interacting with an environment. The agent receives rewards or penalties for its actions and adjusts its decision-making strategy accordingly.

One of the main uses of Reinforcement Learning is in game development. If you want to further understand this branch of Machine Learning we recommend watching this video of a Multi-Agent Hide and Seek game created by OpenAI where the Blue agents need to hide from the Red agents. Each time the Blues get caught, they get a punishment, and each time they hide successfully they get a reward. Without being explicitly programmed how to hide, Blues find a new way each time to hide, create shelters, and block Reds from catching them. The same applies to the Reds who each time find a new way to catch the Blues. This is a great example to illustrate the logic of Reinforcement Learning.

Setting up your environment for your first steps.

4.1 Programming Language

Since we declared that Artificial Intelligence is any application that makes the machine mimic human behavior, we basically need to communicate with the machine. For ML, we will need a programming language.

Many Programming Languages are available today for AI development such as Python, R, Java, C++ …

For us, we will choose Python as our primary programming language for several reasons:

- Python has an extremely simple syntax making it one of the most beginner-friendly programming languages. It's the perfect choice for beginners and who are new to the AI field.
- Python has a wide range of AI & Data libraries and frameworks such as TensorFlow, PyTorch, and Scikit-Learn that provide huge numbers of tools for tasks such as data processing, model training, and performance evaluation.
- Python comes with a lot of versatility as it is used for various types of AI and ML applications, from the most simple data analysis to the most complex deep learning models and tasks. This versatility makes it a great choice for your projects of all sizes and complexity.

- Python has a large community with active members who may help you whenever you run into a problem.
- Interoperability: Python can be easily integrated with other programming languages and tools, making it a great choice for building end-to-end AI solutions.

For now, you can start by installing any distribution of Python on your local PC. We recommend installing Anaconda. Then you need to set up a development environment. For that, we need to install Jupyter Notebook which is a web application for creating and sharing computational documents. In Jupyter Notebook you can create code and text cells to make your code looks more clear and more organized.

One alternative option instead of installing Jupyter Notebook, you can work on Google Colaboratory which is a free Jupyter Notebook Environment that runs on the cloud and is offered by Google.

For this book, we'll suppose you only know the most basic Python programming: data types, data structures, loops, functions, and OOP.

If you don't have any previous coding experience with Python, we recommend learning Python basics before moving to the next parts. We provided you with a link to a Python 3 course we recommend.

4.2 Numpy:

Numpy is a Python library for scientific computing and working on multi-dimensional arrays. It comes with a new

data structure called Numpy Arrays which are very close and similar to Python lists.

It may seem that in Python we can create multi-dimensional lists, but in reality, it's lists within lists, like we did in this example.

```
python_list = [[1,2,3],[4,5,6]]
print(python_list)
```

```
[[1, 2, 3], [4, 5, 6]]
```

We can also create a Numpy array this way:

```
import numpy as np #We start by importing Numpy
numpy_array = np.array([[1,2,3],[4,5,6]])
print(numpy_array)
```

```
[[1 2 3]
 [4 5 6]]
```

You might ask "Why should I use Numpy arrays since I already have Python Lists?". Of course, there should be reasons for that. Numpy may be useful over Lists in some cases for several reasons:

- Numpy is much faster than Lists in all the mathematical operations and calculations.
- Numpy provides a wide range of functions that are not predefined in Python Lists. Numpy has functions for matrix multiplication, reshaping arrays, random array generation, linear algebra operations …

- Numpy arrays have a defined shape, which can be either one or multi-dimensional, while lists in Python are one-dimensional and have no defined shape.

Please also note that while Python Lists can have elements from different types, NumPy arrays can contain only one data type which should be the same for all the elements of the array.

When working on an ML project, we will deal with a large amount of data, so time and memory efficiency is a crucial requirement. For that, we will tend to use Numpy arrays more than Python Lists whenever it's needed.

4.3 Vectorization:

Python is well known as an excellent language for data processing and exploration thanks to its simple syntax. However, this ease of use comes with a downside. Python is much slower in calculations when compared to other programming languages (like C).

Numpy comes here as a savior offering a process called **vectorization** to speed things up. Vectorization is a process that speeds up calculations using NumPy. It's the process of applying an operation to an array, rather than applying it to each element of the array individually.

To better understand the situation let's work on a real example:

```
import time
import numpy as np

# generate a large array
a = np.random.randint(0, 100, 10000000)
start = time.time()

b = np.empty(len(a))
for i in range(len(a)):
    b[i] = a[i] ** 2

end = time.time()
print("Time without vectorization:", end - start)
```

```
Time without vectorization: 5.059529781341553
```

In this example, we created 2 NumPy arrays:

- a: a one-dimensional array containing 10000000 elements with random numbers from 0 to 99.
- b: an empty array having the same shape as array a. Our goal is that array b contains the square of each element in a.

For this method, we used a for loop to traverse the array element by element. We used the time module in Python to give us the execution time.
The execution time for the code above is nearly 5.6 sec.

We tried to approach our problem with a different solution using vectorization:

```
import time
import numpy as np

# generate a large array
a = np.random.randint(0, 100, 10000000)

start = time.time()

b = a ** 2

end = time.time()
print("Time with vectorization:", end - start)
```

```
Time with vectorization: 0.017594099044799805
```

In this example, instead of traversing the array element per element using a for loop, we used vectorization by multiplying the vector a by itself.

The execution time for the second approach is 0.0176 sec which is nearly ***300 times*** faster than the first non-vectorization approach.

The NumPy built-in functions will help us a lot in working on vectorized solutions and approaches.

4.4 Pandas:

While NumPy is a highly efficient and fast array manipulation library, it does not have that amount of flexibility and tools for handling built-in data structures. Here comes our friend Pandas who's built on top of NumPy to fulfill those limitations. Pandas is a Python library that

provides us with 2 built-in data structures for data manipulations. The big amount of useful functions Pandas has, makes it highly efficient in many steps of the ML life cycle such as the data preprocessing task.

The two built-in Pandas data structures are: Pandas Series and Pandas DataFrames

4.4.1 Series:

Pandas Series: are one-dimensional arrays with axis labels. We can create a Pandas Series this way:

```
import pandas as pd

# create a series from a list
s = pd.Series([1, 3, 5, 10, 6, 8])
print(s)
```

```
0     1
1     3
2     5
3    10
4     6
5     8
dtype: int64
```

Each element of the Series is associated with a default label (index). We still can manipulate those labels and input our custom indexes using the index argument the following way:

```
import pandas as pd

# create a series from a list
s = pd.Series([1, 3, 5, 10, 6, 8],
              index = ['Banana','Apple','Peach','Strawberries','Orange','Fig'])
print(s)
```

```
Banana           1
Apple            3
Peach            5
Strawberries    10
Orange           6
Fig              8
dtype: int64
```

You might ask what's the difference then between this Pandas Series and the Python dictionaries since both deal with 'index-value' pairs as you can see. Here are the main differences:

- Dictionaries is a Python default data structure that allows us to store key-value pairs. It offers some built-in methods to manipulate your data. Dictionaries in Python are unordered data structures.
- Series are **one-dimensional arrays** with axis-labels. It offers a vast amount of methods and attributes quite different, for the most part, from those of a python dictionary. In addition, Pandas Series is an ordered data structure.

4.4.2 DataFrame:

Pandas DataFrame is a two-dimensional labeled data structure. It's very similar to an Excel Spreadsheet, containing columns and rows.

There are many ways to create a Pandas DataFrame: from python dictionaries, from Numpy arrays, from CSV files, from Pandas Series …

```
import pandas as pd
#Creating a dictionary
countries = {'Country': ['USA', 'India', 'China', 'Japan'],
             'Population': [331002651, 1369000000, 1403500365, 126529100],
             'Capital': ['Washington, D.C.', 'New Delhi', 'Beijing', 'Tokyo'],
             'Continent': ['North America', 'Asia', 'Asia', 'Asia']}
df = pd.DataFrame(countries) #Creating a DataFrame from a dictionary
df
```

	Country	Population	Capital	Continent
0	USA	331002651	Washington, D.C.	North America
1	India	1369000000	New Delhi	Asia
2	China	1403500365	Beijing	Asia
3	Japan	126529100	Tokyo	Asia

In this example, the dictionary keys are the column indexes in the DataFrame we named df. Each column aside is based on a Pandas Series containing elements of the same type. Those elements are indexed by both their column index (column name) and their row index (default index from 0 to 3 of Pandas Series, and of course we can change them if we want to).

For now, you can note that Pandas DataFrame is a data structure containing columns and rows. One row of data is called an observation, and one column is called a feature.

DataFrames come with a lot of built-in functions to help us with many data tasks from selecting and indexing data, cleaning and preprocessing data, grouping and aggregating data, merging and joining data, filtering and transforming data, plotting data…

We covered all those functions you may need in the next part of the book where we will work on everything with data: from the basic data explorations tasks to data pre-processing and exploratory data analysis tasks before moving to the classical Machine Learning algorithms.